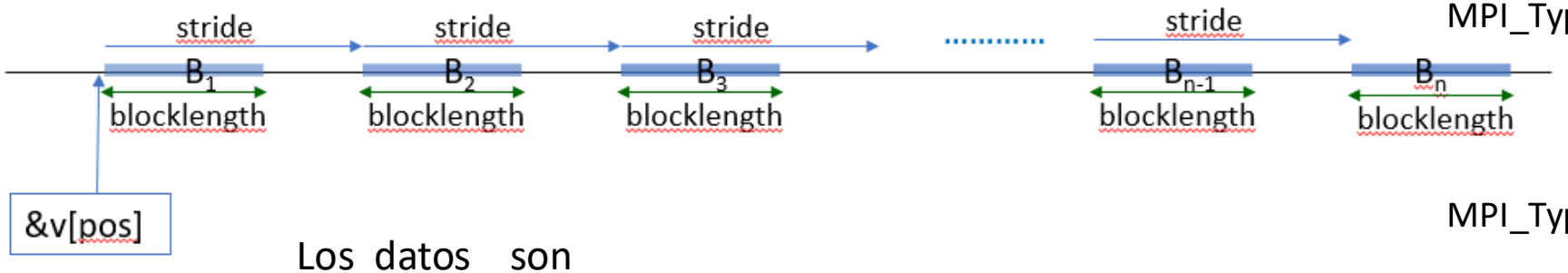
Transparencias MPI

Ejercicios Tipos Derivados

Dado un array, queremos enviar en un solo mensaje un conjunto de bloques de igual longitud y con la misma separación:



MPI_Type_vector(int n, int blocklength, int stride,

MPI_Datatype oldtype,

MPI_Datatype *newtype)

MPI_Type_commit(MPI_Datatype * newtype)

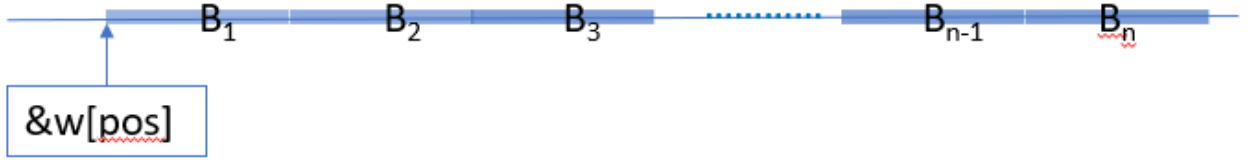
Los datos son empaquetados en el envío del mensaje:

MPI_Send(&v[pos], 1, newtype,…….)

$$t_c = t_s + \mathrm{n}\cdot\mathrm{blocklength}\cdot t_w$$

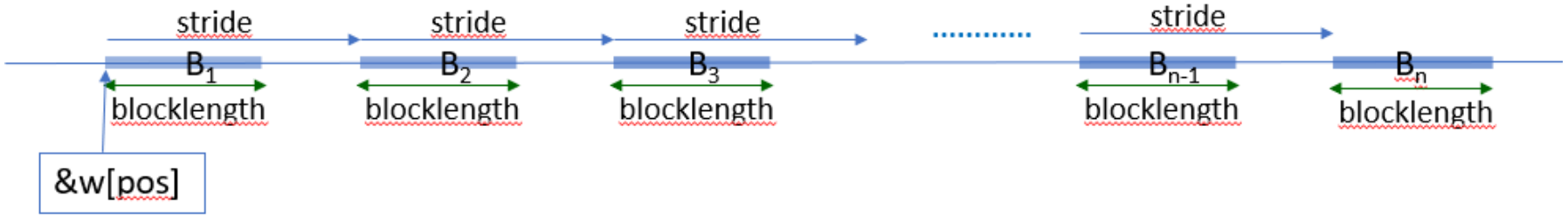
En el proceso receptor se pueden almacenar empaquetados:

MPI_Recv(&w[pos], n*blocklength, oldtype,…….)

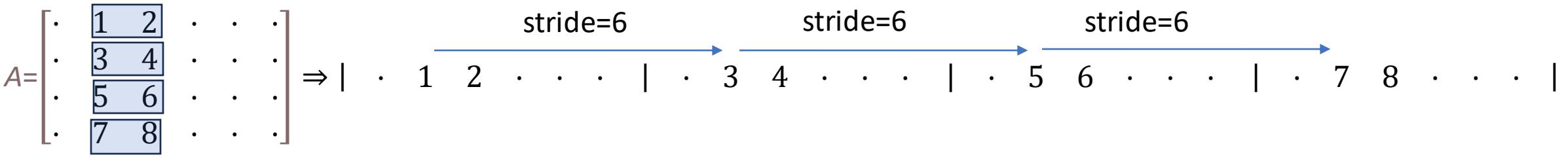


o con la misma estructura que el dato almacenado en el proceso emisor:

MPI_Recv(&w[pos], 1, newtype,…….)



Ejemplo 1




```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

count=4
blocklength=2
stride=6
oldtype=MPI_INT

MPI_Send(&A[0][1], 1, newtype,…….). En el contenido del mensaje solo se encuentran los bloques: 1 2 3 4 5 6 7 8

MPI_Recv(&B[0][2], 1, newtype,…….). Los bloques se desempaquetan y se almacenan con arreglo al tipo derivado definido:

$$B=\begin{bmatrix} \cdot & \cdot & 1 & 2 & \cdot & \cdot \\ \cdot & \cdot & 3 & 4 & \cdot & \cdot \\ \cdot & \cdot & 5 & 6 & \cdot & \cdot \\ \cdot & \cdot & 7 & 8 & \cdot & \cdot \end{bmatrix}$$

MPI_Recv(&v[0], 8, MPI_INT,…….). El mensaje recibido se almacena directamente en la variable:

$$v = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8]$$

Ejemplo 2

$$A = \begin{bmatrix} \boxed{1} & \cdot & \cdot & \cdot \\ \cdot & \boxed{2} & \cdot & \cdot \\ \cdot & \cdot & \boxed{3} & \cdot \\ \cdot & \cdot & \cdot & \boxed{4} \end{bmatrix} \Rightarrow A = [1 \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad 2 \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad 3 \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad 4]$$

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

count=4
blocklength=1
stride=5
oldtype=MPI_INT

MPI_Send(&A[0][0], 1, newtype,…….). En el contenido del mensaje solo se encuentran los bloques de un elemento: $1 \quad 2 \quad 3 \quad 4$

MPI_Recv(&B[0][0], 1, newtype,…….). Los bloques se desempaquetan y se almacenan con arreglo al tipo derivado definido:

$$B = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 2 & \cdot & \cdot \\ \cdot & \cdot & 3 & \cdot \\ \cdot & \cdot & \cdot & 4 \end{bmatrix}$$

MPI_Recv(&v[0], 4, MPI_INT,…….). El mensaje recibido se almacena directamente en la variable:

v=$[1 \quad 2 \quad 3 \quad 4]$

Ejemplo 3

$$A = \begin{bmatrix} \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 2 & \cdot \\ \cdot & 3 & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot \end{bmatrix} \Rightarrow A = [\cdot \quad \cdot \quad \cdot \quad 1 \quad \cdot \quad \cdot \quad 2 \quad \cdot \quad \cdot \quad 3 \quad \cdot \quad \cdot \quad 4 \quad \cdot \quad \cdot \quad \cdot]$$

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

count=4
blocklength=1
stride=3
oldtype=MPI_INT

MPI_Send(&A[0][3], 1, newtype,…….). En el contenido del mensaje solo se encuentran los bloques de un elemento: $1 \quad 2 \quad 3 \quad 4$

MPI_Recv(&B[0][3], 1, newtype,…….). Los bloques se desempaquetan y se almacenan con arreglo al tipo derivado definido:

$$B = \begin{bmatrix} \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 2 & \cdot \\ \cdot & 3 & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot \end{bmatrix}$$

MPI_Recv(&v[0], 4, MPI_INT,…….). El mensaje recibido se almacena directamente en la variable:

v=$[1 \quad 2 \quad 3 \quad 4]$

Ejercicio 1

- Se quiere optimizar el reparto de matrices de dimensión 3nx3n, donde n es un número natural mayor o igual a 1, cuya estructura se presenta a continuación:

$$A = \begin{pmatrix} A_0 & 0_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & A_1 & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} & A_2 \end{pmatrix} \in ¡^{3n\times 3n}, \quad A_i \in ¡^{n\times n}, \quad 0_{n\times n} = \text{matriz nula}$$

- Suponiendo que se tienen 3 procesos, implementa un programa paralelo MPI que cumpla las siguientes condiciones:
  - P0 utiliza la función leeMat para obtener la matriz **A**: A=leeMat(N).
  - P0 debe repartir dicha matriz en matrices locales **Al** entre tres procesos de manera que P_0 se quede con A_0, P_1 se quede con A_1 y P_2 se quede con A_2, usando para ello tipos derivados
- Calcula el tiempo de comunicaciones
- Ejemplo n=2:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 5 & 0 & 0 \\ 0 & 0 & 6 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 0 & 10 & 11 \end{pmatrix} \Rightarrow \begin{array}{l} Al(P_0) = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \\ Al(P_1) = \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} \\ Al(P_2) = \begin{pmatrix} 8 & 9 \\ 10 & 11 \end{pmatrix} \end{array}$$

Ejercicio 1

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

$$A = \left(\begin{array}{c|c|c} A_0 & 0_{n\times n} & 0_{n\times n} \\ \hline 0_{n\times n} & A_1 & 0_{n\times n} \\ \hline 0_{n\times n} & 0_{n\times n} & A_2 \end{array}\right)$$

$$A_0 = \begin{pmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{pmatrix} \quad 0_{n\times n} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

$$A_1 = \begin{pmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{pmatrix}$$

$$A_2 = \begin{pmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{pmatrix}$$

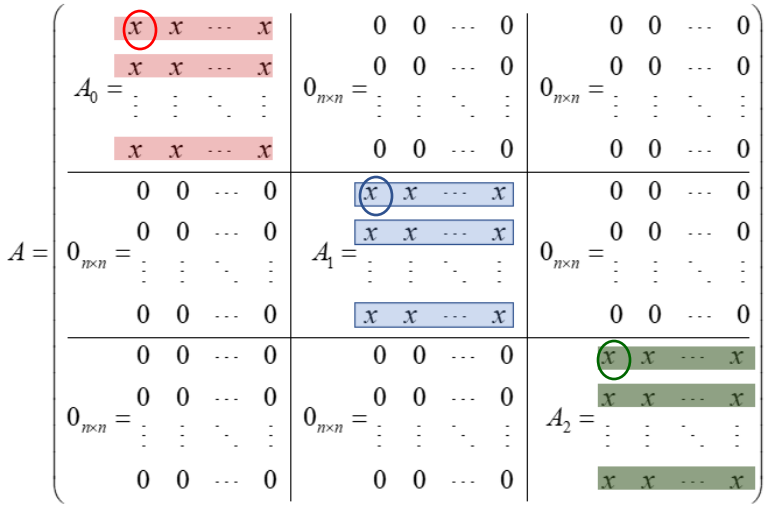
Definir un nuevo tipo para enviar eficientemente A_i, i= 1, 2

count=n
blocklength=n
stride=3n

```
MPI_Datatype newtype;
MPI_Type_vector(n, n, 3*n, MPI_DOUBLE, &newtype);
```

Ejercicio 1

P_0 reparte la matriz **A** en matrices locales **Al** entre los tres procesos, de manera que P_0 se queda con A_0, P_1 se queda con A_1 y P_2 se queda con A_2, usando el tipo derivado anterior:



```
double max_mat(double A[3*n][3*n], Al[n][n], int p, int ip){
  MPI_Status stat;
  MPI_Datatype ntype;
  MPI_Type_vector(n, n, 3*n, MPI_DOUBLE, &ntype);
  MPI_Type_commit(&ntype);
  if ( ip == 0 ){
      MPI_Send(&A[n][n],1, ntype, 1, 100, MPI_COMM_WORLD);
      MPI_Send(&A[2*n][2*n],1, ntype, 2, 100, MPI_COMM_WORLD);
      MPI_Sendrecv(A, 1, ntype, 0, 100,  Al, n*n, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &stat);
  }
  else
      MPI_Recv(Al, n*n, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &stat);
  MPI_Type_free (&ntype);
}
```

$$t_c = 2\left(t_s + n^2 t_w\right)$$

Ejercicio 2 (Parcial 2020)

Se quiere implementar en MPI el envío desde el proceso 0 al resto de procesos de la diagonal principal y anti-diagonal de una matriz cuadrada ***A*** de dimensión **N**, empleando para ello tipos de datos derivados (uno para cada tipo de diagonal), con la menor cantidad posible de mensajes. Supondremos que:

- ***N*** es una constante conocida.
- Los elementos de la diagonal principal son: $A_{00}, A_{11}, A_{22}, \ldots, A_{N-1,N-1}$.
- Los elementos de la anti-diagonal son: $A_{0,N-1}, A_{1,N-2}, A_{2,N-3}, \ldots, A_{N-1,0}$.
- Solo el proceso 0 posee la matriz *A* y enviará la totalidad de dichas diagonales al resto de procesos.

Un ejemplo para una matriz de tamaño $N = 5$ sería:

$$A = \begin{pmatrix} * & & & & * \\ & * & & * & \\ & & * & & \\ & * & & * & \\ * & & & & * \end{pmatrix}$$

(a) Completa la siguiente función, donde los procesos del 1 en adelante almacenarán sobre la matriz A las diagonales recibidas:

void sendrecv_diagonals(double A[N][N])

```
int MPI_Bcast(void *buf, int count,
    MPI_Datatype datatype, int root,
    MPI_Comm comm)
```

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

(a) **Solución**: Definiremos un tipo de datos para envíos de la diagonal principal y otro para envíos de la antidiagonal. A continuación, realizaríamos difusiones con ambos tipos de datos:

```
void sendrecv_diagonals(double A[N][N]) {
  MPI_Datatype principal,antidiag;
  MPI_Type_vector(N, 1, N+1, MPI_DOUBLE, &principal);
  MPI_Type_commit(& principal);
  MPI_Type_vector(N, 1, N-1, MPI_DOUBLE, &antidiag);
  MPI_Type_commit(&antidiag);
  MPI_Bcast(A, 1, principal, 0, MPI_COMM_WORLD);
  MPI_Bcast(&A[0][N-1], 1, antidiag, 0, MPI_COMM_WORLD);
  MPI_Type_free(&principal);
  MPI_Type_free(&antidiag);
}
```

$$A = \begin{pmatrix} * & & & * \\ & * & * & \\ & & * & \\ & * & & * \\ * & & & * \end{pmatrix}$$

Diagonal principal:
- count=N
- blocklength=1
- stride=N+1

Antidiagonal:
- count=N
- blocklength=1
- stride=N-1

Ejercicio 2 (Parcial 2020)

(b) Completa esta otra función, variante de la anterior, donde todos los procesos (incluido el proceso 0) almacenarán sobre los vectores **prin** y **anti** las correspondientes diagonales:

```
void sendrecv_diagonals(double A[N][N], double prin[N], double anti[N]) {
  int id;
  MPI_Datatype principal, antidiag;
  MPI_Comm_rank(MPI_COMM_WORLD, &id);
  if (id==0) {
    MPI_Type_vector(N, 1, N+1, MPI_DOUBLE, &principal);
    MPI_Type_commit(& principal);
    MPI_Type_vector(N, 1, N-1, MPI_DOUBLE, &antidiag);
    MPI_Type_commit(&antidiag);
    MPI_Sendrecv(A, 1, principal, 0, 10, prin, N , MPI_DOUBLE, 0, 10, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Sendrecv(&A[0][N-1], 1, antidiag, 0, 20, anti, N, MPI_DOUBLE, 0, 20, MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    MPI_Type_free(&principal);
    MPI_Type_free(&antidiag);
  }
  MPI_Bcast(prin, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  MPI_Bcast(anti, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
```

```
int MPI_Sendrecv(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, int
    dest, int sendtag, void *recvbuf, int
    recvcount, MPI_Datatype recvtype, int
    source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

```
int MPI_Bcast(void *buf, int count,
    MPI_Datatype datatype, int root,
    MPI_Comm comm)
```

Ejercicio 3 (Final 2020)

Queremos repartir una matriz de *M* filas y *N* columnas que se encuentra en el proceso 0 entre 4 procesos mediante un reparto por columnas cíclico. Como ejemplo, se muestra el caso de la siguiente matriz de 6 filas y 8 columnas:

$$A(P_0) = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \end{bmatrix}$$

$$Al(P_0) = \begin{bmatrix} 1 & 5 \\ 9 & 13 \\ 17 & 21 \\ 25 & 29 \\ 33 & 37 \\ 41 & 45 \end{bmatrix} \quad Al(P_1) = \begin{bmatrix} 2 & 6 \\ 10 & 16 \\ 18 & 26 \\ 26 & 36 \\ 34 & 46 \\ 42 & 56 \end{bmatrix} \quad Al(P_2) = \begin{bmatrix} 3 & 7 \\ 11 & 15 \\ 19 & 23 \\ 27 & 31 \\ 35 & 39 \\ 43 & 47 \end{bmatrix} \quad Al(P_3) = \begin{bmatrix} 4 & 8 \\ 12 & 16 \\ 20 & 24 \\ 28 & 32 \\ 36 & 40 \\ 44 & 48 \end{bmatrix}$$

Ejercicio 3 (Final 2020)

Implementa una función en MPI que realice, mediante primitivas punto a punto y de la forma más eficiente posible, el envío y recepción de dicha **matriz**. Nota: La recepción de la matriz deberá hacerse en una matriz compacta (en Al), como muestra el ejemplo anterior. **Nota**: El número de columnas se asume que es un múltiplo de 4 y se reparte siempre entre 4 procesos. Para la implementación se recomienda utilizar la siguiente cabecera:

int MPI_Reparte_col_cic(float A[M][N], float Al[M][N/4])

Solución:

Se trataría de definir un nuevo tipo de datos derivado, de manera que se envíen en un solo mensaje los elementos que le corresponden a cada proceso.

Ejercicio 3 (Final 2020)

Por ejemplo, a P1 se le enviarían los elementos coloreados en rojo (A matriz con M=6 filas y N=8 columnas):

$$A(P_0) = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \end{bmatrix}$$

Como las filas de la matriz se almacenan en posiciones consecutivas de memoria, se trata de un reparto cíclico de elementos almacenados en posiciones consecutivas entre los cuatro procesos.


```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

count= M*N/4
blocklength=1
stride=4

Una vez se define este nuevo tipo de dato, se realizarían envíos desde las posiciones 0, 1, 2, o 3 de ese nuevo tipo de dato. En recepción, los datos se almacenarían en matrices de tamaño M filas y N/4 columnas; es decir, se recibirían como datos almacenados en posiciones consecutivas.

Ejercicio 3 (Final 2020)

```
int MPI_Reparte_col_cic(float A[M][N], float Al[M][N/4]){
  int id, i;
  MPI_Datatype col;
  MPI_Comm_rank(MPI_COMM_WORLD, &id);
  MPI_Type_vector(M*N/4, 1, 4, MPI_FLOAT ,&col);
  MPI_Type_commit(&col);
  if (id==0) {
    for (i=1;i<4;i++)
      MPI_Send(&A[0][i], 1, col, i, 10 , MPI_COMM_WORLD);
    MPI_Sendrecv(A, 1, col, 0, 10, Al, M*N/4, MPI_FLOAT, 0, 10, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }
  else
    MPI_Recv(Al, M*N/4, MPI_FLOAT, 0, 10, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  MPI_Type_free(&col);
  return 0;
}
```

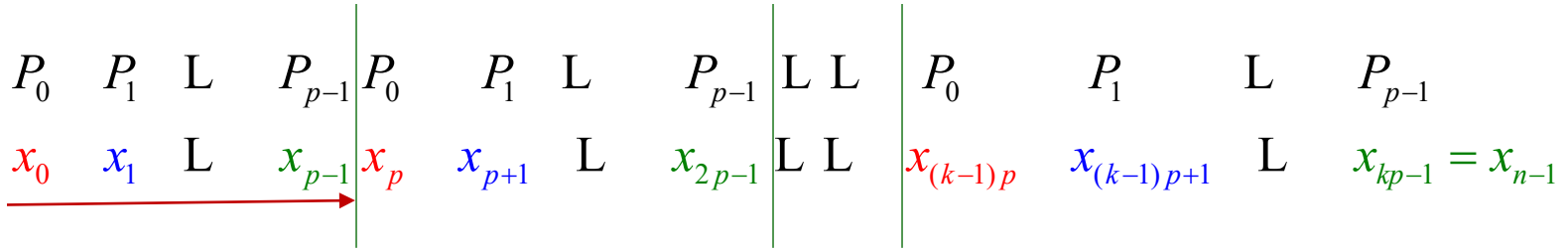
```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)

int MPI_Sendrecv(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, int
    dest, int sendtag, void *recvbuf, int
    recvcount, MPI_Datatype recvtype, int
    source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

Ejercicio 4

Se quiere implementar una función en MPI que permita repartir de forma cíclica un vector **x** de dimensión **n** entre **p** procesos, con la siguiente cabecera:

void reparto_ciclico(double x[], double xl[], int n)

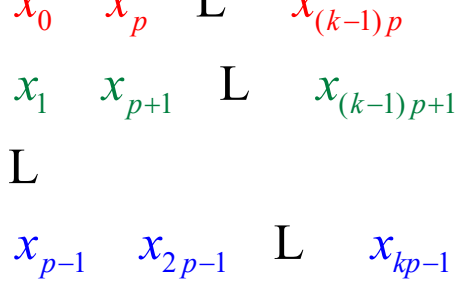
siendo **xl** la parte local una vez se ha repartido **x**. Impleméntala definiendo para ello un tipo derivado que permita realizar comunicaciones eficientes (suponer que **n** es divisible entre el número de procesos **p**). Calcula el tiempo de comunicaciones. ¿Cuál sería el tiempo de comunicaciones si no se hubiera usado el tipo derivado?



$$k = \frac{n}{p}$$

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

Queremos en un solo envío, empaquetar y enviar los elementos del mismo color



count=k
blocklength=1
stride=p

```
MPI_Datatype ciclico ;
MPI_Type_vector(k, 1, p, MPI_DOUBLE, &ciclico);
MPI_Type_commit(&ciclico);
```

Ejercicio 4

P_0 debe enviar a cada P_i los elementos del mismo color que le corresponden
Cada P_i debe recibir en el vector local xl las componentes que le corresponden

$$x(P_0): \begin{array}{cccccccccccccc} P_0 & P_1 & \mathrm{L} & P_{p-1} & P_0 & P_1 & \mathrm{L} & P_{p-1} & \mathrm{L} & \mathrm{L} & P_0 & P_1 & \mathrm{L} & P_{p-1} \\ x_0 & x_1 & \mathrm{L} & x_{p-1} & x_p & x_{p+1} & \mathrm{L} & x_{2p-1} & \mathrm{L} & \mathrm{L} & x_{(k-1)p} & x_{(k-1)p+1} & \mathrm{L} & x_{kp-1} = x_{n-1} \end{array}$$

$$\begin{array}{l} xl(P_0): x_0 \quad x_p \quad \mathrm{L} \quad x_{(k-1)p} \\ xl(P_1): x_1 \quad x_{p+1} \quad \mathrm{L} \quad x_{(k-1)p+1} \\ \mathrm{M} \\ xl(P_{p-1}): x_{p-1} \quad x_{2p-1} \quad \mathrm{L} \quad x_{kp-1} \end{array}$$

Ejercicio 4

P_0	P_1	L	P_{p-1}	P_0	P_1	L	P_{p-1}	L L	P_0	P_1	L	P_{p-1}
x_0	x_1	L	x_{p-1}	x_p	x_{p+1}	L	x_{2p-1}	L L	$x_{(k-1)p}$	$x_{(k-1)p+1}$	L	$x_{kp-1} = x_{n-1}$

```
void reparto_ciclico(double x[], double xl[], int n){
int p, id, k, i;
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &id);
k=n/p;
MPI_Datatype ciclico;
MPI_Status stat;
MPI_Type_vector(k, 1, p, MPI_DOUBLE, &ciclico);
MPI_Type_commit(&ciclico);
if (id==0) {
  for (i=1; i<p; i ++)
    MPI_Send(&x[i], 1, ciclico, i, 10, MPI_COMM_WORLD);
  MPI_Sendrecv(&x[0], 1, ciclico, 0, 10, &xl[0], k, MPI_DOUBLE, 0, 10, MPI_COMM_WORLD, &stat);
}
else
  MPI_Recv(&xl[0], k, MPI_DOUBLE, 0, 10, MPI_COMM_WORLD, &stat);
MPI_Type_free(&ciclico);
}
```

```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)
```

Receive a message from one process:

```
int MPI_Recv(void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)
```

Combined send and receive:

```
int MPI_Sendrecv(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, int
    dest, int sendtag, void *recvbuf, int
    recvcount, MPI_Datatype recvtype, int
    source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

p-1 mensajes de k=n/p datos:

$$t_c = (p-1)\left(t_s + \frac{n}{p} t_w\right)$$

Sin tipos derivados:

$$t_c = \left(n - \frac{n}{p}\right)(t_s + t_w)$$

Ejercicio 5

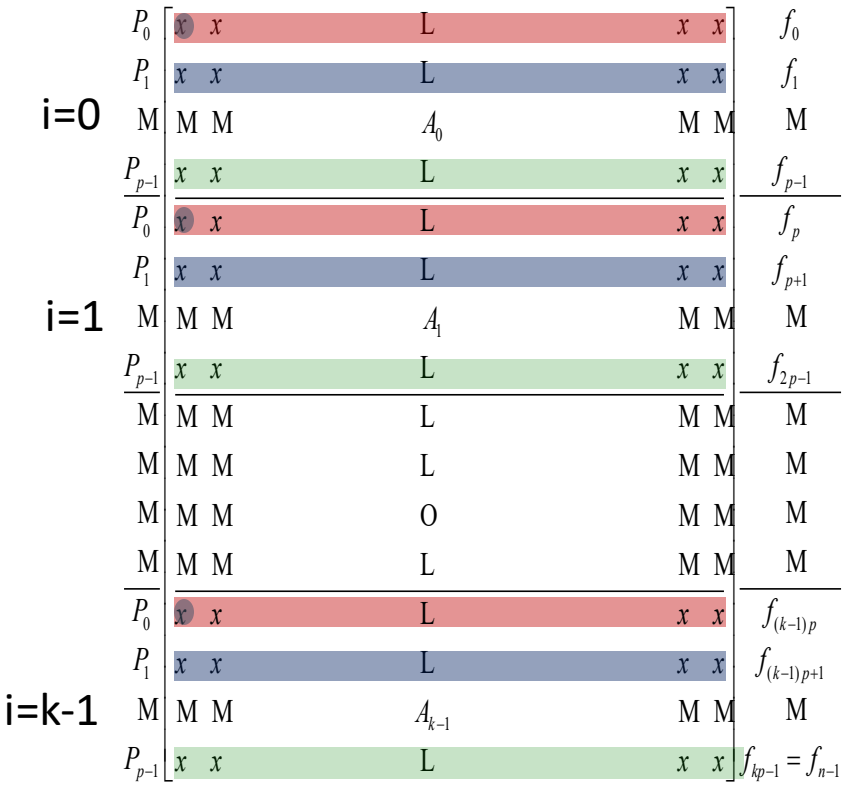
Dada una matriz cuadrada **A** de dimensión **n** almacenada en el proceso P0, implementa el código necesario para que **A** sea repartida cíclicamente por filas entre todos los procesos, almacenándolas en las matrices locales **Al**

a) usando MPI_Scatter (como en la sesión 4 de la práctica 3)

b) usando MPI_Type_vector

calculando en ambos casos el tiempo de comunicaciones ¿Cuál tendrá menor tiempo paralelo?

$$A(P_0) \in \mathbb{R}^{n \times n}$$



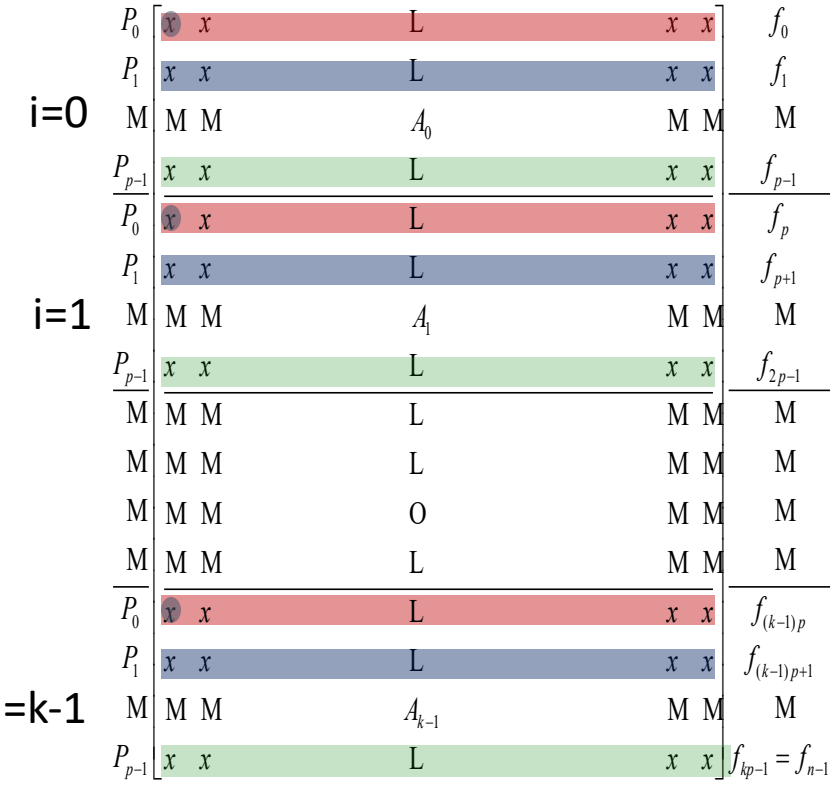
Ejercicio 5-a)

Para i=0, 1, 2, …, k-1

Reparto de A_i (MPI_Scatter): una fila para cada matriz Al de cada uno de los procesos

Fin para

$A(P_0) \in \mathbb{R}^{n \times n}$



$$Al(P_0) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_0 \\ \bar{f}_1 = fp \\ \vdots \\ \bar{f}_{k-1} = f_{(k-1)p} \end{matrix}$$

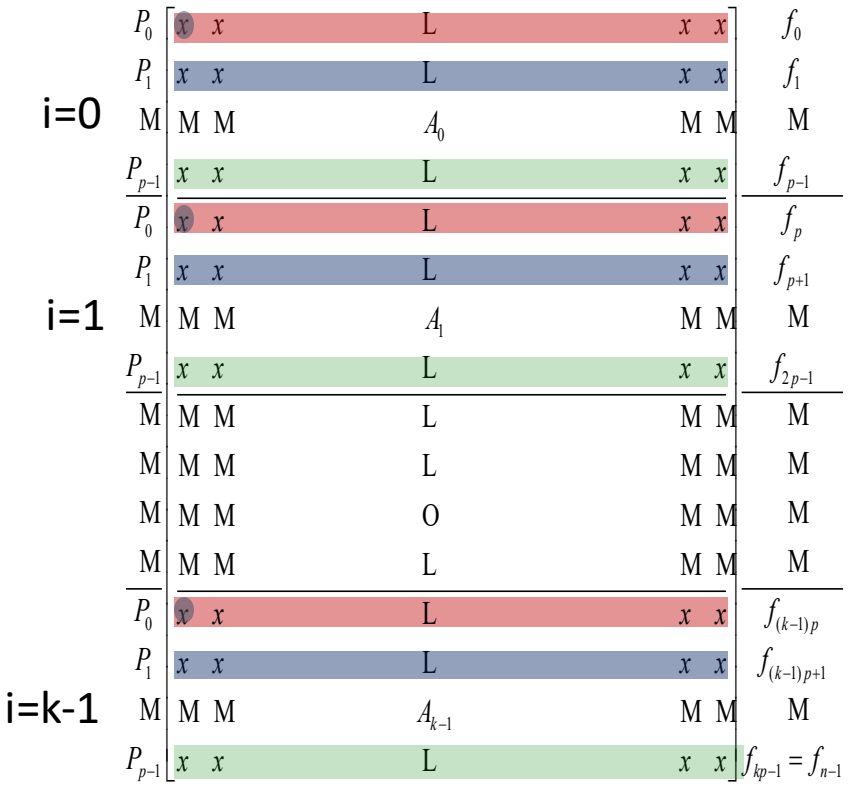
$$Al(P_1) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_1 \\ \bar{f}_1 = f_{p+1} \\ \vdots \\ \bar{f}_{k-1} = f_{(k-1)p+1} \end{matrix}$$

$$\vdots$$

$$Al(P_{p-1}) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_{p-1} \\ \bar{f}_1 = f_{2p-1} \\ \vdots \\ \bar{f}_{k-1} = f_{kp-1} \end{matrix}$$

Ejercicio 5-a)

$A \in \mathbb{R}^{n \times n}$



$$Al(P_0) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_0 \\ \bar{f}_1 = fp \\ \vdots \\ \bar{f}_{k-1} = f_{(k-1)p} \end{matrix}$$

$$Al(P_1) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_1 \\ \bar{f}_1 = f_{p+1} \\ \vdots \\ \bar{f}_{k-1} = f_{(k-1)p+1} \end{matrix}$$

$$\vdots$$

$$Al(P_{p-1}) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_{p-1} \\ \bar{f}_1 = f_{2p-1} \\ \vdots \\ \bar{f}_{k-1} = f_{kp-1} \end{matrix}$$

Para i=0 hasta k-1

Reparto de A_i

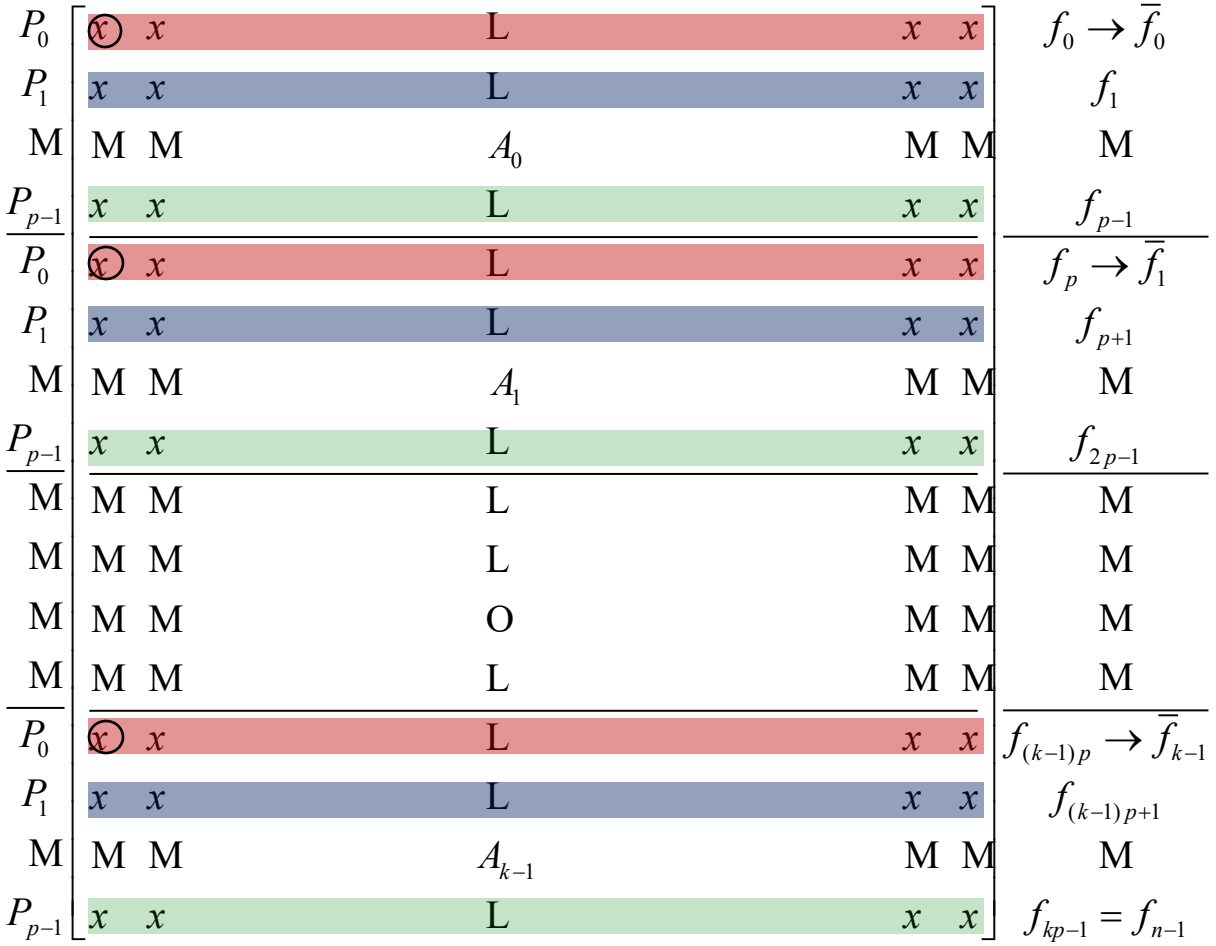
Fin para

i	Primer elemento de A	Primer elemento de Al
0	A[0][0]	Al[0][0]
1	A[p][0]	Al[1][0]
2	A[2*p][0]	Al[2][0]
...	...	...
k-1	A[(k-1)*p][0]	Al[k-1][0]

```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

Ejercicio 5-a)

$A \in \mathbb{R}^{n \times n}$



f_i fila i-ésima de la matriz A

$\bar{f}_i$ fila i-ésima de la matriz Al

```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

i	Primer elemento de A	Primer elemento de Al
0	A[0][0]	Al[0][0]
1	A[p][0]	Al[1][0]
2	A[2*p][0]	Al[2][0]
...	...	...
k-1	A[(k-1)*p][0]	Al[k-1][0]

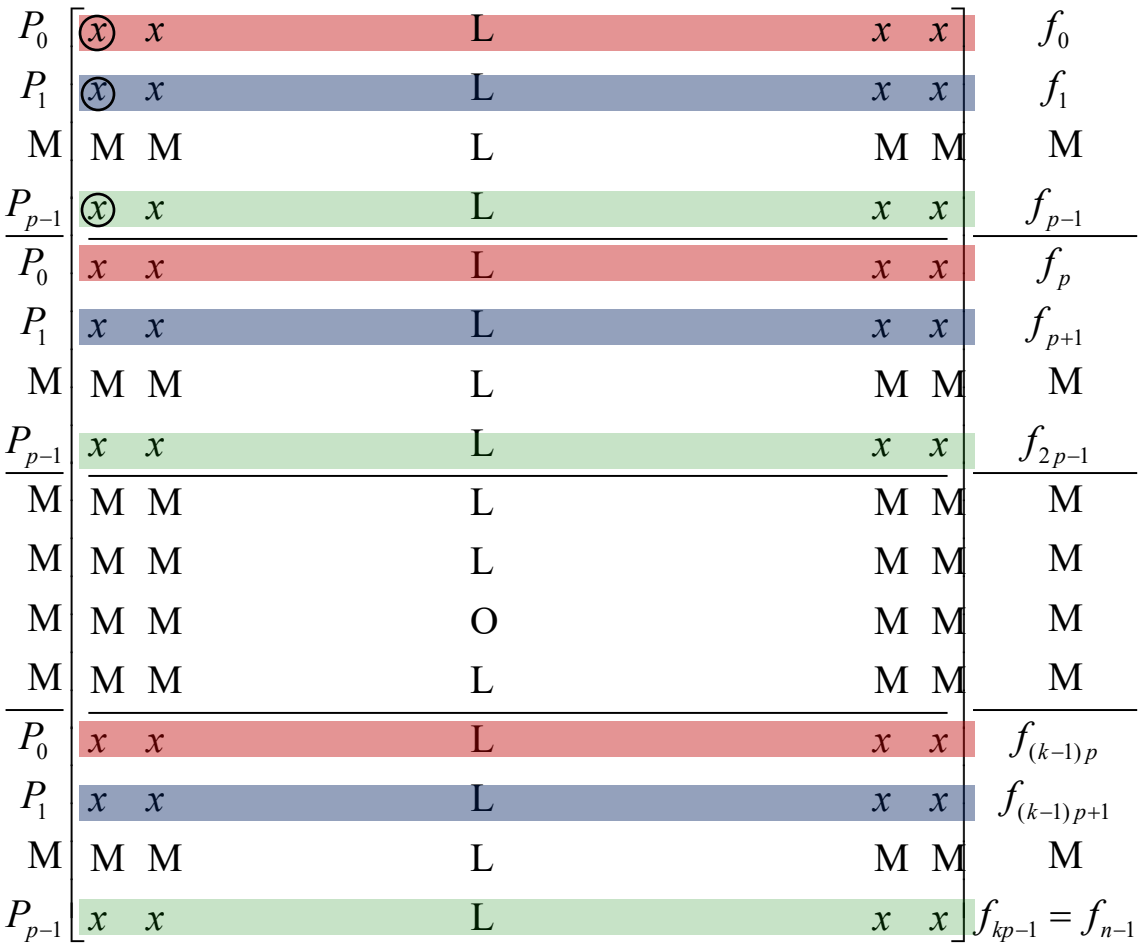
```
double A[n][n], Al[n][n];
Int i, p, k;
MPI_Comm_size(MPI_COMM_WORLD, &p);
k=n/p;
for(i=0; i<k; i++)
   MPI_Scatter(&A[i*p][0], n, MPI_DOUBLE, &Al[i][0], n,
                    MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

$$t_c = k(p-1)\left(t_s + nt_w\right) = \frac{n(p-1)}{p}\left(t_s + nt_w\right)$$

Ejercicio 5-b)

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

count=k=n/p blocklength=n stride=p*n




```
double A[n][n], Al[n][n];
Int i, p, ip, k;
MPI_Status   stat;
MPI_Comm_size(MPI_COMM_WORLD,&p);
MPI_Comm_rank(MPI_COMM_WORLD,&ip);
k=n/p;
MPI_Datatype ntype;
MPI_Type_vector(k, n, p*n, MPI_DOUBLE, &ntype);
MPI_Type_commit(&ntype);
if (ip==0) {
   for(i=1; i<p; i++)
      MPI_Send(&A[i][0], 1, ntype, i, 100, MPI_COMM_WORLD);
   MPI_Sendrecv(A, 1, ntype, 0, 100, Al, k*n, MPI_DOUBLE, 0, 100,
  }                              MPI_COMM_WORLD, &stat);
else
   MPI_Recv(Al, k*n, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &stat);
MPI_Type_free(& ntype);
```

Al crear el nuevo tipo de datos, en un solo mensaje se envían los datos del mismo color

$$t_c = (p-1)\left(t_s + \frac{n^2}{p} t_w\right)$$

Ejercicio 5-b)

$$\text{MPI\_Scatter: } t_c = \frac{n(p-1)}{p}\left(t_s + nt_w\right) = \frac{n(p-1)}{p}t_s + \frac{n^2(p-1)}{p}t_w$$

$$\text{Type\_Vector: } t_c = (p-1)\left(t_s + \frac{n^2}{p}t_w\right) = (p-1)t_s + \frac{n^2(p-1)}{p}t_w$$

- Las dos distribuciones tienen el mismo coeficiente de t_w
- Veamos qué ocurre con los coeficientes de t_s

$$\frac{n(p-1)}{p} > (p-1) \leftrightarrow n > p$$

- Luego si $n > p$, entonces el tiempo paralelo es mayor para el reparto mediante MPI_Scatter

Ejercicio 6

Se quiere implementar una función en MPI que permita repartir de forma cíclica las columnas de una matriz **A** de dimensión **mxn** entre **p** procesos, con la siguiente cabecera:

void reparto_ciclico_col(double A**, double Al*, int m, int n)

siendo **Al** la matriz local una vez se ha repartido **A**. Impleméntala definiendo para ello un tipo derivado que permita realizar comunicaciones eficientes (suponer que **n** es divisible entre el número de procesos **p**). Calcula el tiempo de comunicaciones. Ejemplo con m=4, n=6 y p=3 procesos:

$$A(P_0) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix}$$

$$Al(P_0) = \begin{pmatrix} 0 & 3 \\ 6 & 9 \\ 12 & 15 \\ 18 & 21 \end{pmatrix}$$

$$Al(P_1) = \begin{pmatrix} 1 & 4 \\ 7 & 10 \\ 13 & 16 \\ 19 & 22 \end{pmatrix}$$

$$Al(P_2) = \begin{pmatrix} 2 & 5 \\ 8 & 11 \\ 14 & 17 \\ 20 & 23 \end{pmatrix}$$

Ejercicio 6

$m = 4, n = 6, p = 3, k = \frac{n}{p} = \frac{6}{3} = 2$

$$A(P_0) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix} \rightarrow Al(P_0) = \begin{pmatrix} 0 & 3 \\ 6 & 9 \\ 12 & 15 \\ 18 & 21 \end{pmatrix} \quad Al(P_1) = \begin{pmatrix} 1 & 4 \\ 7 & 10 \\ 13 & 16 \\ 19 & 22 \end{pmatrix} \quad Al(P_2) = \begin{pmatrix} 2 & 5 \\ 8 & 11 \\ 14 & 19 \\ 20 & 25 \end{pmatrix}$$

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

En este ejemplo:

count= $2x4 = 8$

blocklength $= 1$

stride $= 3$

En general:

count= $km = \frac{mn}{p}$

blocklength $= 1$

stride $= p$

```
MPI_Datatype ciclico ;
MPI_Type_vector(k*m, 1, p, MPI_DOUBLE, &ciclico);
MPI_Type_commit(&ciclico);
```

Ejercicio 6

$$A(P_0) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix}$$

```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)
```

Receive a message from one process:

```
int MPI_Recv(void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)
```

Combined send and receive:

```
int MPI_Sendrecv(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, int
    dest, int sendtag, void *recvbuf, int
    recvcount, MPI_Datatype recvtype, int
    source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

```
void reparto_ciclico_col(double A**, double Al*, int m, int n) {
int p, id, k, j;
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &id);
k=n/p;
MPI_Datatype ciclico;
MPI_Status stat;
MPI_Type_vector(k*m, 1, p, MPI_DOUBLE, &ciclico);
MPI_Type_commit(&ciclico);
if (id==0) {
  for (j=1; j<p; i ++)
    MPI_Send(&A[0][j], 1, ciclico, j, 10, MPI_COMM_WORLD);
  MPI_Sendrecv(&A[0][0], 1, ciclico, 0, 10, &Al[0][0], k*m, MPI_DOUBLE, 0, 10, MPI_COMM_WORLD, &stat);
}
else
  MPI_Recv(&Al[0][0], k*m, MPI_DOUBLE, 0, 10, MPI_COMM_WORLD, &stat);
MPI_Type_free(&ciclico);
}
```

p-1 mensajes de k=m*n/p datos:

$$t_c = (p-1)\left(t_s + \frac{mn}{p} t_w\right)$$

Ejercicio 7

Se quiere implementar una función en MPI que permita repartir de forma por bloques las columnas de una matriz **A** de dimensión **mxn** entre **p** procesos, con la siguiente cabecera:

void reparto_bloques_col(double A**, double Al*, int m, int n)

siendo **Al** la matriz local una vez se ha repartido **A**. Impleméntala definiendo para ello un tipo derivado que permita realizar comunicaciones eficientes (suponer que **n** es divisible entre el número de procesos **p**). Calcula el tiempo de comunicaciones. Ejemplo con m=4, n=6 y p=3 procesos:

$$A(P_0) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix}$$

$$Al(P_0) = \begin{pmatrix} 0 & 1 \\ 6 & 7 \\ 12 & 13 \\ 18 & 19 \end{pmatrix}$$

$$Al(P_1) = \begin{pmatrix} 2 & 3 \\ 8 & 9 \\ 14 & 15 \\ 20 & 21 \end{pmatrix}$$

$$Al(P_2) = \begin{pmatrix} 4 & 5 \\ 10 & 11 \\ 16 & 17 \\ 22 & 23 \end{pmatrix}$$

Ejercicio 7

$m = 4, n = 6, p = 3, k = \frac{n}{p} = \frac{6}{3} = 2$

$$A(P_0) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix} \rightarrow Al(P_0) = \begin{pmatrix} 0 & 1 \\ 6 & 7 \\ 12 & 13 \\ 18 & 19 \end{pmatrix} \quad Al(P_1) = \begin{pmatrix} 2 & 3 \\ 8 & 9 \\ 14 & 15 \\ 20 & 21 \end{pmatrix} \quad Al(P_2) = \begin{pmatrix} 4 & 5 \\ 10 & 11 \\ 16 & 17 \\ 22 & 23 \end{pmatrix}$$

```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

En este ejemplo:

count= 4

blocklength = 2

stride = 6

En general:

count= m

blocklength = k

stride = n

```
MPI_Datatype bloc ;
MPI_Type_vector(m, k, n, MPI_DOUBLE, &bloc);
MPI_Type_commit(&bloc);
```

Ejercicio 7

$$A(P_0) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix}$$

```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)
```

Receive a message from one process:

```
int MPI_Recv(void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)
```

Combined send and receive:

```
int MPI_Sendrecv(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, int
    dest, int sendtag, void *recvbuf, int
    recvcount, MPI_Datatype recvtype, int
    source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

```
void reparto_bloques_col (double A**, double Al*, int m, int n){
int p, id, k, j;
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &id);
k=n/p;
MPI_Datatype bloc;
MPI_Status stat;
MPI_Type_vector(m, k, n, MPI_DOUBLE, &ciclico);
MPI_Type_commit(&bloc);
if (id==0) {
  for (j=1; j<p; i ++)
    MPI_Send(&A[0][j*k], 1, bloc, j, 10, MPI_COMM_WORLD);
  MPI_Sendrecv(&A[0][0], 1, bloc, 0, 10, &Al[0][0], k*m, MPI_DOUBLE, 0, 10, MPI_COMM_WORLD, &stat);
}
else
  MPI_Recv(&Al[0][0], k*m, MPI_DOUBLE, 0, 10, MPI_COMM_WORLD, &stat);
MPI_Type_free(&bloc);
}
```

p-1 mensajes de k=m*n/p datos:

$$t_c = (p-1)\left(t_s + \frac{mn}{p} t_w\right)$$